Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

Implantação e avaliação de técnicas de inspeção de software

Fernando Melim Hottum

JUIZ DE FORA
FEVEREIRO, 2022

Implantação e avaliação de técnicas de inspeção de software

Fernando Melim Hottum

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Sistemas de Informação

Orientador: Marco Antônio Pereira Araújo

JUIZ DE FORA

FEVEREIRO, 2022

Implantação e avaliação de técnicas de inspeção de software

Fernando Melim Hottum

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Marco Antônio Pereira Araújo
Doutor em Engenharia de Sistemas e Computação/UFRJ

Alessandreia Marta de Oliveira Julio
Doutora em Computação - IC/UFF

José Maria Nazar David
Doutor em Engenharia de Sistemas e Computação/UFRJ

JUIZ DE FORA
18 DE FEVEREIRO, 2022

Resumo

A inspeção de *software* é uma revisão feita em artefatos de *software* com a finalidade de encontrar defeitos nos mesmos. Esse tipo de revisão surge da necessidade de implementar *softwares* desenvolvidos em um tempo especificado, garantindo uma menor quantidade de defeitos, reduzindo o custo do desenvolvimento e mantendo a qualidade como prioridade. O objetivo desse trabalho é avaliar a utilização de técnicas existentes para a realização da inspeção de documentos de *software* e verificar se a utilização delas impacta positivamente um time de desenvolvimento que utiliza uma metodologia ágil. As técnicas utilizadas para isso foram a *Perspective Based Reading* e a *Checklist Based Reading*. Essas técnicas foram utilizadas de forma conjunta para tentar reduzir o *backlog* de uma empresa de desenvolvimento de *software*, seguindo a ideia de inspeção de *tickets* para isso. Através de uma análise estatística feita com os dados obtidos após a implantação de tais técnicas, não foi possível observar uma diminuição significativa do *backlog* do time de desenvolvimento escolhido, mas foi possível identificar um novo problema referente à documentação do software que é mantido através dos *tickets* que foram analisados pelas técnicas escolhidas.

Palavras-chave: Inspeção de *software*; desenvolvimento de *software*; técnicas de inspeção; artefatos de *software*.

Abstract

Software inspection is a review performed on software artifacts in order to find defects in them. This type of review arises from the need to implement software developed in a specified time, ensuring the least amount of defects, reducing the cost of development and keeping quality as a priority. The main of this work is to evaluate the use of existing techniques for the inspection of software documents and to verify if the use of those techniques positively impacts a software development team that uses an agile methodology. The techniques used for this were the Perspective Based Reading and Checklist Based Reading. These techniques were used together to try to reduce the backlog of a software development company, following the idea of tickets inspection for this. Through a statistical analysis made with the data obtained after the implementation of such techniques, it was not possible to observe a significant decrease in the backlog of the chosen development team, but it was possible to identify a new problem regarding the documentation of the software that is maintained through the tickets that were analyzed by the chosen techniques.

Keywords: software inspection, software development, inspection techniques, software artifacts, software quality.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

CBR	*Checklist-Based Reading*
GQM	*Goal Question Metric*
PBR	*Perspective-Based Reading*
PICOC	*Population*, *Intervention*, *Comparison*, *Outcome*, *Context*.
UFJF	Universidade Federal de Juiz de Fora
UML	*Unified Modeling Language*

1 Introdução

Em um ambiente empresarial de desenvolvimento de *software*, é de extrema importância que existam processos bem definidos para o desenvolvimento ocorrer como o planejado. Esses processos têm a finalidade de garantir que o artefato a ser desenvolvido será entregue no tempo estimado e com a qualidade garantida. A inspeção de *software* surge justamente para ajudar a cumprir os objetivos citados. Através dela, é definido um processo de revisão de artefatos de *software* que tem como objetivo encontrar defeitos nos mesmos. Sendo um processo de teste estático, onde não é necessário que o código-fonte esteja sendo executado, Fagan (1976) explica que a inspeção envolve os desenvolvedores do produto em um processo formal de investigação que detecta um número maior de defeitos do que um teste de máquina.

Tendo em vista que a inspeção de *software* é um processo que visa melhorar o desenvolvimento em vários aspectos, foram desenvolvidas técnicas específicas para que a inspeção seja formalizada e padronizada. As técnicas utilizadas no estudo experimental documentado nesta monografia foram:

- *Perspective-Based Reading*(PBR): documentada inicialmente por Shull (1998), representa um conjunto de técnicas de leitura elaboradas para detectar defeitos em documentos de requisitos que foram escritos em linguagem natural. Pode ser utilizada também para a inspeção de código-fonte. No estudo experimental documentado nesta monografia, essa técnica foi utilizada levando em consideração a perspectiva de um analista de requisitos de um time de desenvolvimento;

- *Checklist-Based Reading*(CBR): utilizada primeiramente por Fagan (1976), é uma técnica onde o artefato a ser inspecionado será avaliado através de uma lista de perguntas que devem ser respondidas. A avaliação do artefato é feita de acordo com as respostas obtidas.

Foi identificada a necessidade da utilização dessas técnicas em uma empresa de desenvolvimento de *software*, que segue uma abordagem ágil de desenvolvimento, pois ela

não utiliza uma forma padrão para analisar os problemas documentados por seus clientes. Essa empresa mantém um software jurídico através de um sistema de cadastro de *tickets*, que representam um problema documentado por um cliente. Inicialmente, esse problema é analisado por uma equipe de suporte ao produto, que decide se de fato é um problema no sistema, direcionando o *ticket* aos desenvolvedores, ou se é apenas uma interpretação errada sobre o funcionamento do sistema por parte do cliente que cadastrou o *ticket*. O grande problema em não utilizar uma forma padrão para analisar os *tickets* nessa empresa é que eles estão sendo analisados pela equipe de suporte seguindo uma informalidade que pode estar impactando negativamente o trabalho dos desenvolvedores. Com isso, muitos dos *tickets* que são encaminhados para a equipe de desenvolvedores são recusados pela mesma. Os desenvolvedores alegam que cerca de, aproximadamente, 21% dos *tickets* enviados a eles não documentam um defeito no sistema, mas sim um comportamento esperado e especificado.

Ter uma equipe de desenvolvimento de *software* analisando 21% dos *tickets* que não deveriam ter sido direcionados a ela gera um problema grave para a empresa, pois seus funcionários estão analisando itens que não fazem parte de seu escopo de trabalho, afetando negativamente o desempenho dos mesmos e gerando um prejuízo para a empresa. Além disso, os clientes que criaram os *tickets* esperam que o problema que foi reportado seja consertado pois, pelo tempo de espera do *feedback* da empresa, eles presumem que de fato seja um problema no código-fonte e não um comportamento esperado do sistema. Quando recebem um *feedback* comunicando que o comportamento reportado não é proveniente de um erro no código-fonte mas sim um comportamento esperado, a confiança na empresa e no produto diminui devido ao tempo de espera para ter um *feedback* que não era o esperado.

A importância de se utilizar técnicas de inspeção de *software* em empresas que lidam com desenvolvimento ágil pode ser um desafio, já que muitas das técnicas foram desenvolvidas, inicialmente, em um contexto onde o conceito de desenvolvimento ágil ainda não era extremamente disseminado como hoje. A falta do uso dessas técnicas para formalizar o processo de inspeção dos artefatos de *software* da empresa, utilizada como um estudo de caso, pode estar causando um retrabalho enorme nas análises de *tickets*.

O estudo experimental documentado nesta monografia existe para documentar uma análise feita sobre a utilização de técnicas de inspeção de *software* em um contexto onde o desenvolvimento ágil é adotado como padrão, onde foi analisado se a utilização dessas técnicas teve um impacto positivo.

Para avaliar se as técnicas utilizadas de fato tiveram um impacto significativo na empresa, foram feitas análises estatísticas, que estão melhor documentadas no Capítulo 4, seguindo o raciocínio de se utilizar testes de hipótese. Essas análises foram realizadas de acordo com o dados referentes ao número de *tickets* cadastrados e recusados pela equipe de desenvolvimento que está sendo utilizada no experimento. Esses dados foram fornecidos pela empresa que gere essa equipe de desenvolvedores.

1.1 Objetivos

1.1.1 Objetivo Geral

Observando os problemas reportados, as técnicas de inspeção de *software* citadas anteriormente, PBR e CBR, foram implantadas para auxiliar a empresa a ter uma maior assertividade em suas análises, diminuindo a quantidade de *tickets* que são encaminhados erroneamente para a equipe responsável pelo desenvolvimento e correções do produto. Dessa forma, a satisfação do cliente melhoraria, pois ele receberia um *feedback* com uma análise mais precisa. Além disso, a produtividade da equipe de desenvolvimento também seria maior, pois a quantidade de itens em seu *backlog* que não fazem parte de seu escopo diminuiria, o que resulta da diminuição de análises de requisitos desnecessárias por profissionais especializados em desenvolvimento de *software*.

A equipe de desenvolvimento que será impactada pelo uso das técnicas PBR e CBR utiliza o SCRUM, definido por Schwaber e Beedle (2002), como metodologia ágil de desenvolvimento de software. Tendo em vista que as técnicas PBR e CBR foram desenvolvidas em um cenário onde metodologias ágeis de desenvolvimento de software não eram muito utilizadas, o desafio desse estudo é utilizar essas técnicas para verificar se elas podem trazer benefícios a equipes de desenvolvimento de software que seguem uma abordagem ágil de desenvolvimento.

1.1.2 Objetivo Específico

Para avaliar o impacto que a implantação das técnicas de inspeção têm na produtividade em um time de desenvolvimento de *software*, a métrica utilizada será o número de *tickets* que foram cadastrados no *backlog* do time mas que não documentavam um problema no produto, e sim um comportamento especificado e esperado. Será feita uma comparação da quantidade de *tickets* cadastrados em um cenário onde as técnicas de inspeção não foram implantadas com um cenário onde foram.

1.2 Organização do trabalho

Essa monografia está dividida em cinco capítulos. O Capítulo 2 documenta a revisão sistemática feita para encontrar trabalhos que fazem parte do mesmo escopo que esse estudo experimental, assim como o referencial teórico e os estudos utilizados. O Capítulo 3 aborda a metodologia utilizada para a realização do experimento. O Capítulo 4 expõe os resultados encontrados a partir da análise estatística feita, documentada no Capítulo 3, e o Capítulo 5 é referente à conclusão obtida do experimento realizado.

2 Planejamento da Revisão Sistemática

Uma revisão sistemática tem como objetivo avaliar e interpretar todos os tipos de estudos disponíveis e relevantes para o tópico que está sendo pesquisado, segundo Kitchenham (2004). Através dela, é possível fazer uma avaliação rigorosa das fontes de informações que serão utilizadas em um estudo, verificando sua confiabilidade e veracidade. Nesse estudo experimental, esse tipo de revisão foi utilizada para prover conhecimentos relacionados às técnicas de inspeção de software e como elas podem ser utilizadas para melhorar a análise de artefatos de *software*.

Além da revisão, existe também o mapeamento sistemático, que tem um objetivo diferente da revisão. Esse mapeamento visa estruturar o tema da pesquisa, sem responder perguntas específicas com profundidade (DERMEVAL; COELHO; BITTENCOURT, 2019), como é feito na revisão sistemática. Ele procura exibir a quantidade e frequência de publicações existentes dentro de uma determinada área (GARCIA; SILVA; NASCIMENTO, 2018). Nesse estudo, foi utilizada apenas a revisão sistemática, pois a área de atuação desse experimento já estava bem definida. O necessário para dar continuidade a esse experimento era encontrar estudos disponíveis de acordo com as restrições que serão explicitadas nas próximas seções.

Nesse capítulo, a fundamentação teórica será explicitada na seção 2.1 e a revisão sistemática será nas seções posteriores. No final da revisão sistemática, serão explicitados os artigos resultantes da mesma que foram utilizados para selecionar as técnicas utilizadas nesse trabalho.

2.1 Fundamentação Teórica

As técnicas de inspeção de *software* são utilizadas para auxiliar na detecção de defeitos em artefatos de *software* (GAZERANI et al., 2015). Elas padronizam a análise a ser feita, o que auxilia a examinar minuciosamente os artefatos desejados. Geralmente essas técnicas estão associadas a um processo de inspeção de *software*, que inicialmente foi definido por

Fagan (1976), mas sua utilização não precisa depender de um processo pré-definido ou de uma reunião de inspeção, pois isso não contribui para o número de defeitos encontrados (JOHNSON, 1998). Elas podem ser utilizadas de forma a complementar um processo já existente na organização que deseja utilizá-las. Nesse contexto, as técnicas PBR e CBR foram escolhidas para serem utilizadas nesse estudo experimental de forma a integrar os processos já definidos na organização em que estão sendo implementadas. Essas técnicas foram escolhidas pois os artigos resultantes da revisão sistemática documentada na Seção 2.4 demonstram bons resultados quando essas técnicas são utilizadas para inspecionar artefatos de *software* escritos em linguagem natural.

2.1.1 Taxonomia para especificar defeitos

Como o estudo experimental documentado nesta monografia tem como finalidade avaliar a utilização de técnicas de inspeção de *software* para identificar defeitos, é necessário que os tipos de defeitos que podem existir sejam definidos. Shull (1998) define esses defeitos da seguinte forma:

- omissão: classificação voltada para especificar problemas no artefato analisado que são considerados defeitos pelo fato de faltar especificações mais detalhadas no mesmo;
- ambiguidade: classificação dada ao defeito encontrado em um artefato que documenta requisito(s) de forma ambígua, levando o leitor a ter múltiplas interpretações de um mesmo requisito;
- inconsistência: classificação dada a um defeito encontrado em um artefato que documenta requisitos conflitantes;
- fato incorreto: classificação dada a um defeito encontrado em um artefato que documenta requisito(s) que não são verdadeiros;
- informação estranha: classificação dada a um defeito encontrado em um artefato que documenta requisito(s) que possui informações desnecessárias para a análise do mesmo;
- outros: qualquer outro tipo de defeito encontrado se enquadra nessa classificação.

A necessidade de definir classificações, como as apresentadas, para os principais tipos de defeitos existentes é imprescindível, porque é necessário ter um conhecimento prévio do que está sendo procurado durante a inspeção.

2.1.2 PBR

Segundo Shull (1998), a PBR define um conjunto de técnicas de leitura que pode auxiliar seus utilizadores a detectarem defeitos em artefatos de *software* que documentam requisitos. Esse conjunto de procedimentos tem como finalidade revisar o artefato escolhido assumindo o ponto de vista de um *stakeholder* que utilizará o que está documentado nesse artefato (SHULL, 1998). A seguinte diferenciação para técnica, método e documento de *software* é adotada por Shull (1998):

- técnica: um série de passos, que devem ser executados em ordem, que são utilizados para completar uma tarefa;
- método: uma descrição que visa definir quando e como as técnicas devem ser aplicadas, verificando o contexto e a necessidade delas serem utilizadas;
- documento de *software*: qualquer artefato de *software* que é textual, o que inclui o documento de requisitos.

Nesse contexto, a PBR é definida como um método, pois é através dela que é definida qual técnica pode ser utilizada da forma mais eficaz dentro de um contexto previamente especificado (SHULL, 1998). A PBR foi escolhida para ser utilizada nesse estudo pois o uso dela visa identificar defeitos em documentos de *software* que estão sendo analisados individualmente (SHULL, 1998), e é exatamente isso o que é necessário para realizar as análises de *tickets* por parte do analista da organização utilizada como caso de estudo.

2.1.3 CBR

A utilização de *checklists* para identificar defeitos em *software* foi inicialmente proposta por Fagan (1976), que criou a técnica CBR que utiliza perguntas e respostas para declarar

se um artefato de *software* possui algum defeito ou não. Utilizando a definição da Seção 2.1.2, a CBR, utilizada primeiramente por Fagan (1976), pode ser considerada uma técnica pelo fato de descrever uma série de perguntas que devem ser respondidas em ordem, tendo como resposta apenas "sim"ou "não". Quando uma pergunta obtém um "não"como resposta, significa que o artefato que está sendo inspecionado através da *checklist* possui um defeito.

Na Figura 2.1 é expressado o raciocínio utilizado para se escolher a CBR como técnica para realizar a inspeção nos artefatos, mostrando alguns dos elementos que serão analisados e o que é preciso analisar neles. Tendo os artefatos a serem inspecionados e algumas das informações que necessitam ser analisadas exibidas na Figura 2.1, é possível identificar qual técnica pode ser utilizada para analisá-los. A CBR foi escolhida pois as informações contidas nos artefatos especificados podem ser analisadas através de uma série de perguntas para verificar se essas informações estão presentes e consistentes. Os artefatos e dados que foram utilizados para realizar esse estudo experimental serão melhor explicados no Capítulo 3.

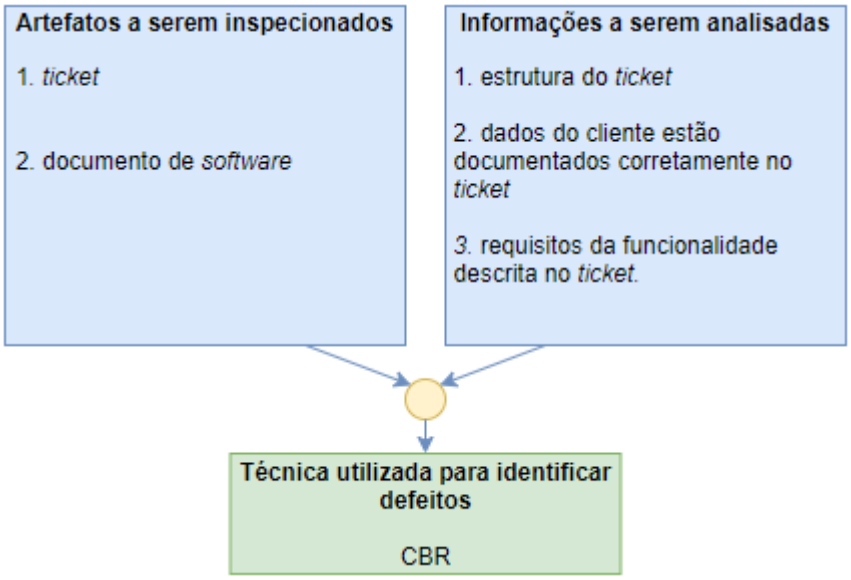

Figura 2.1: Raciocínio utilizado para escolher a CBR

2.2 Mapeamento da pesquisa das técnicas de inspeção de *software*

Antes da revisão sistemática ser definida de fato, é importante definir como as técnicas de inspeção de software foram mapeadas para serem escolhidas e explicitar o modelo adotado para fazer isso. Essa etapa é importante, pois o uso do modelo adotado leva à criação de perguntas que serão respondidas pela revisão sistemática.

2.2.1 Modelo GQM

As metas e métricas utlizadas para definir quais técnicas serão utilizadas foram estipuladas utilizando a abordagem GQM(do inglês, *Goal Question Metric*), definida por Caldiera e Rombach (1994). Essa abordagem parte do pressuposto que, para se atingir as metas estipuladas, é necessário que perguntas sejam respondidas. Para as respostas dessas perguntas serem obtidas, é necessário definir algumas métricas.

A Figura 2.2 exibe a estrutura hierárquica definida pelo modelo GQM. Através dessa estrutura, pode-se observar que, no topo da hierarquia, estão presentes as metas. Através dessas metas, são estipuladas perguntas que estão presentes no segundo nível da hierarquia, cujas respostas são obtidas durante a execução da revisão sistemática. A procura por essas respostas deve ser feita com as métricas definidas, que se encontram no terceiro nível da hierarquia, para cada pergunta criada.

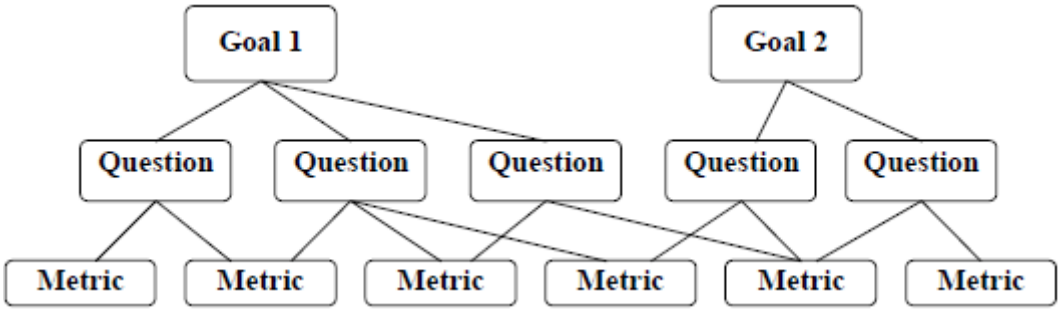

Figura 2.2: Modelo hierárquico do GQM (CALDIERA; ROMBACH, 1994)

2.2.2 Processo GQM

O processo utilizado para definir as metas, perguntas, métricas e obter as respostas do que está sendo procurado foi definido por Caldiera e Rombach (1994), e ele consiste em cinco etapas. As etapas são:

- definição das metas que visam alcançar o(s) objetivo(s) definido(s) do estudo;
- criação de perguntas que visam estabelecer a medição das metas. Tendo todas as perguntas respondidas, deve-se ter a capacidade de verificar se as metas foram alcançadas;
- estipular as métricas necessárias para responder as perguntas definidas na etapa anterior;
- estipular quais são os mecanismos de coleta de dados;
- coletar os dados, utilizando os mecanismos definidos na etapa anterior, e analisá-los.

2.2.3 Definição de metas, perguntas e métricas

Seguindo os três primeiros passos do processo definido na seção anterior, foram definidas a seguinte meta, pergunta e métricas:

- meta 1(G1): definir as técnicas de inspeção de software que serão utilizadas.
  - pergunta 1(Q1): quais técnicas podem ser utilizadas a partir da perspectiva de um analista de requisitos?
    - métrica 1(M1): poderão ser escolhidas apenas duas técnicas. A ideia é que o utilizador delas consiga fazer uma análise rápida, o que se torna mais difícil de ser feito se a quantidade de técnicas utilizadas for elevada;
    - métrica 2(M2): as técnicas selecionadas devem ter sido utilizadas em pelo menos um estudo experimental.

A meta foi definida de acordo com os objetivos específicos presentes na Seção 1.1.2, estipulando o conhecimento necessário para o estudo experimental ser realizado.

A pergunta foi criada tendo como objetivo obter o conhecimento necessário para a meta ser atingida. Através das respostas dessa pergunta, foi possível decidir quais foram as técnicas selecionadas. As métricas são utilizadas como uma forma para medir e restringir o conhecimento que deve ser obtido para responder as perguntas associadas a elas.

Por fim, o seguinte objetivo foi definido: **analisar** as métricas definidas para se obter as técnicas de inspeção de software desejadas, **com a finalidade** de utilizar essas técnicas **para verificar** sua eficácia, levando em consideração o **ponto de vista** de um analista de requisitos **inserido em um contexto** de manutenção de *software*. Como a empresa que utilizou as técnicas selecionadas através da revisão sistemática já possui um processo de desenvolvimento sólido, o mesmo não foi levado em consideração para a criação do objetivo.

2.2.4 Definição dos mecanismos de coleta de dados

Seguindo o quarto passo do processo definido na Seção 2.2.2, foi estipulado o principal mecanismo de coleta de dados, que é o Portal de Periódicos do CAPES, acessível através da UFJF(Universidade Federal de Juiz de Fora). Através desse portal, foram acessadas as seguintes bases de dados: *Scopus*, *ScienceDirect*, *IEEExplore* e *Compendex*.

2.3 Estrutura da Revisão Sistemática

Tendo as principais fontes de dados definidas, é importante que a revisão sistemática seja modelada, e é isso o que é feito nessa seção.

2.3.1 Definição das palavras-chave para busca

Através da pergunta definida na Seção 2.2.3, é necessário que a busca que foi feita nas bases de dados escolhidas na Seção 2.2.4 seja especificada. A abordagem PICOC(do inglês, *Population*, *Intervention*, *Comparison*, *Outcome*, *Context*) foi adotada, que, segundo Petticrew e Roberts (2008), é um método que visa descrever os cinco principais elementos de uma pergunta a ser pesquisada, que são:

- população(*population*): responsável por descrever o grupo de pessoas que possuem

interesse na revisão que é feita;

- intervenção(*intervention*): responsável por definir os objetos de estudo;
- comparação(*comparison*): responsável por definir a que os elementos da Intervenção estão sendo comparados;
- resultado(*outcome*): responsável por definir o que está sendo procurado, melhorado ou concluído;
- contexto(*context*): responsável por definir o contexto do estudo.

Esses elementos foram descritos através do uso de palavras-chave, que possuem a finalidade de identificar materiais que pertençam à mesma área de interesse desse estudo.

A Tabela 2.1 exibe as palavras-chave que foram utilizadas na pesquisa que busca responder a pergunta definida na Seção 2.2.3.

Tabela 2.1: PICOC utilizada para responder a pergunta Q1 da meta G1

PICOC	**Palavras-Chave**
Population	*developer*, *manager*, *product owner*, *tester*, *scrum master*
Intervention	*software inspection technique*
Comparison	Não foi utilizado
Outcome	*technique*
Context	*software inspection*, *software maintenance*, *software quality*, *quality assurance*

2.3.2 Definição dos termos de busca

Tendo as palavras-chave definidas utilizando o PICOC na Seção 2.3.1, foi criado um termo de busca para a pergunta definida na Seção 2.2.3. Ele foi criado utilizando os operadores *AND* e *OR*, que são responsáveis por montar a estrutura de interpretação das palavras-chave. A Tabela 2.2 mostra o termo de busca utilizado para fazer a busca referente às palavras-chave definidas na Tabela 2.1.

Tabela 2.2: Termo de busca utilizado para responder a pergunta Q1 da meta G1

(”developer”OR ”manager”OR ”product owner”OR ”scrum master”OR ”tester”) AND (”software inspection technique”) AND (”technique”)

2.3.3 Definição dos critérios de inclusão e exclusão

Tendo a revisão sistemática como principal metodologia adotada para se obter referências bibliográficas nesse estudo, é importante que exista uma forma de filtrar todo o material recebido durante a execução dela (KITCHENHAM, 2004). Dessa forma, foram criados critérios de inclusão e exclusão de acordo com a métrica definida na Seção 2.2.3, conforme pode ser visto na Tabela 2.3. Cada material obtido foi analisado, onde os critérios definidos foram utilizados para decidir se o material seria utilizado ou não.

Tabela 2.3: Critérios de inclusão e exclusão da pergunta Q1 da meta G1

Tipo	**Descrição**
Inclusão	Artigos que documentam estudos experimentais onde técnicas de inspeção de *software* são utilizadas.
Exclusão	Artigos que não podem ser acessados através das bases definidas na seção 2.2.4, utilizadas através do Portal de Periódicos do CAPES. Artigos onde as técnicas de inspeção de *software* utilizadas dependem da análise de diagramas UML ou código-fonte.

2.4 Condução da Revisão sistemática

A execução da revisão sistemática faz parte da quinta etapa do processo definido na Seção 2.2.2, e é isso o que será abordado nessa seção. Para auxiliar na revisão, foi utilizada a ferramenta Parsifal[1]. Essa revisão foi executada em Agosto de 2021.

Um processo foi definido para executar a busca e filtrar os estudos encontrados. Esse processo pode ser visualizado na Figura 2.3, onde é exibida a quantidade de artigos que foram obtidos e excluídos em cada parte da execução do mesmo. Ele foi dividido nos cinco passos a seguir:

- passo 1: obter os artigos de acordo com a *string* de busca definido na Seção 2.3.2;
- passo 2: remover os artigos duplicados que foram obtidos no passo anterior;
- passo 3: filtrar os artigos restantes de acordo com a leitura do título deles;
- passo 4: filtrar os artigos restantes de acordo com a leitura de seus resumos;

[1]Acessível através de ⟨https://parsif.al/⟩

- passo 5: filtrar os artigos restantes de acordo com a leitura inteira deles.

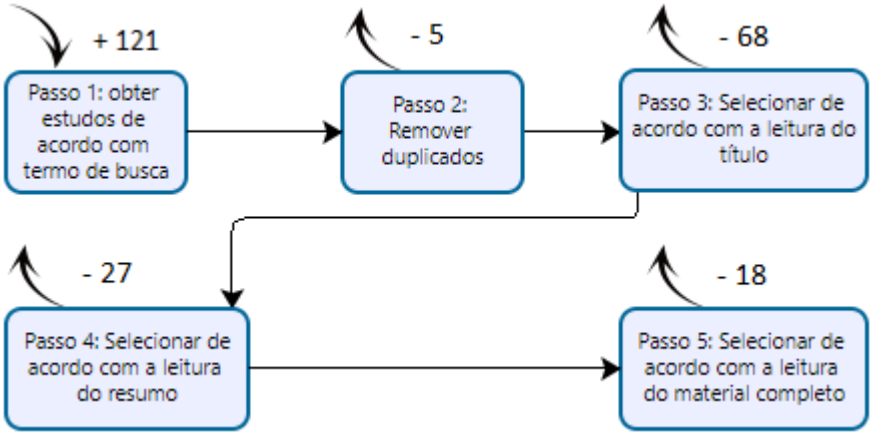

Figura 2.3: Processo utilizado para realizar a revisão sistemática

O primeiro passo foi executado usando a *string* de busca nas bases definidas na Seção 2.2.4. Após a execução dele, foram obtidos cento e vinte e um artigos. Na Figura 2.4 podem ser observados quantos artigos foram obtidos em cada base de dados. Após a execução do segundo passo, foram identificados cinco artigos duplicados. Os passos três, quatro e cinco foram executados levando em consideração os critérios de inclusão e exclusão definidos na Seção 2.3.3. O passo três eliminou sessenta e oito artigos, o passo quatro eliminou vinte e sete e o passo cinco eliminou dezoito, tendo como resultado três artigos utilizados para responder a pergunta definida na Seção 2.2.3.

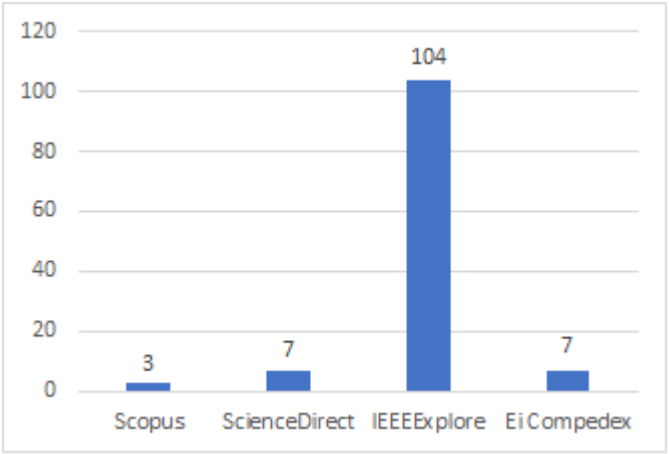

Figura 2.4: Número de artigos encontrados em cada base de dados

Tendo os artigos que foram utilizados selecionados após a execução do quinto passo, foi possível identificar quais técnicas de inspeção de software já foram usadas e

que não dependem da análise de diagramas UML e de código-fonte, de acordo com os critérios de exclusão definidos na Tabela 2.3.3. Através desses artigos, foi fácil identificar os livros ou estudos que documentam a criação dessas técnicas pois eles são referenciados. Dessa forma, as técnicas utilizadas para dar sequência a esse estudo experimental foram selecionadas e documentadas na Seção 2.1, enquanto os estudos anteriores encontrados através da revisão sistemática estão documentados na próxima seção.

2.5 Trabalhos Anteriores

Através da revisão sistemática, foram obtidos os artigos que documentam a utilização de técnicas de inspeção de *software* em um estudo experimental onde o foco era utilizar essas técnicas para analisar documentos de *software*. Nessa seção, serão abordados os estudos obtidos e que foram encontrados através da revisão sistemática.

Biffl (2000) realizou um estudo experimental que teve como objetivo investigar a influência que a utilização da PBR e CBR pode ter na performance do indivíduo que está realizando uma inspeção de acordo com seu conhecimento técnico. O estudo de Biffl (2000) parte do pressuposto que a performance do indivíduo que realiza uma inspeção não é impactada de acordo com o ambiente em que essa inspeção está sendo feita, ou seja, a inspeção realizada pelo indivíduo sozinho pode ser tão boa quanto uma realizada em uma reunião de inspeção.

Biffl (2000) utilizou alunos, que também são desenvolvedores, de uma universidade para usar a PBR e CBR visando identificar defeitos em documentos de *software*. Cada aluno pôde utilizar apenas uma das técnicas apresentadas onde, ao final de cada inspeção, foi entregue uma lista com os defeitos encontrados. Como resultado desse estudo, foi feita uma comparação entre o uso da PBR e CBR, levando em consideração o nível de conhecimento de cada inspetor. A PBR demonstrou ser menos eficiente que a CBR nesse estudo de acordo com o número de defeitos encontrados pelos inspetores. Além disso, a CBR demonstrou ter uma maior cobertura de análise do documento mas tendo um maior tempo gasto para realizar a inspeção.

A principal diferença do estudo experimental de Biffl (2000) com o proposto nesse documento é que a PBR e CBR não foram utilizadas com a finalidade de comparar a efeti-

vidade das duas técnicas, mas sim usadas em conjunto para tentar solucionar um problema de uma empresa. A relevância do estudo de Biffl (2000) para o estudo experimental proposto nesse documento se deve ao fato de ele demonstrar que a CBR pode ser uma técnica eficiente para se encontrar defeitos em documentos de *software* mesmo quando utilizada por inspetores inexperientes que analisam os documentos de forma individual.

Outro estudo encontrado pela revisão sistemática é o criado por McMeekin, Konsky e Chang (2008). No artigo, os autores avaliam a percepção de desenvolvedores novatos enquanto utilizam a CBR. Para isso, utilizaram a taxonomia de Bloom, que foi proposta para ser utilizada como um *framework* para avaliar o conhecimento que desenvolvedores possuem sobre um determinado *software* (BUCKLEY; EXTON, 2003). O estudou tentou definir o nível de percepção que desenvolvedores novatos podem chegar enquanto utilizam a CBR para realizar inspeções de código. Apesar da inspeção ter sido de código-fonte, foram utilizadas *checklists* juntamente com a especificação do sistema.

Apesar do cenário do estudo de McMeekin, Konsky e Chang (2008) diferir do cenário utilizado para realizar o experimento documentado nessa monografia, ele se torna relevante pois mostra que os inspetores que utilizaram a CBR juntamente com a especificação do *software* que utilizaram, tiveram uma percepção maior para encontrar defeitos nos artefatos que analisaram. Nos critérios de exclusão de artigos definidos na Seção 2.3.3 ficou explícito que trabalhos relacionados à inspeção de código-fonte não seriam aceitos para compor o referencial teórico dessa monografia. Esse trabalho foi selecionado pelo fato de ter sido realizado um estudo utilizando inspetores que usaram a CBR juntamente com a documentação do *software* que estava sendo inspecionado, e é exatamente isso que é feito no experimento proposto nessa monografia.

O terceiro artigo encontrado através da revisão sistemática documenta um estudo feito comparando a PBR, CBR, e a *Ad hoc*, que se trata de uma técnica para análise de artefatos de *software* que não segue nenhuma metodologia específica e depende muito do conhecimento prévio que o inspetor possui sobre os artefatos que ele inspeciona (CIOLKOWSKI; LAITENBERGER; BIFFL, 2003). A proposta do estudo feito por Ciolkowski (2009) foi fazer uma análise estatística com os dados resultantes provenientes de diferentes estudos já existentes que buscavam comparar a utilização das técnicas PBR, CBR e *Ad*

hoc. Como resultado da análise documentada no artigo, chegou-se à conclusão que a PBR demonstrou ser mais efetiva do que o método *Ad hoc* ao analisar documentos de *software*, mas menos efetiva que a CBR.

O estudo de Ciolkowski (2009) se torna relevante para esse trabalho pois é comprovado, com base nos dados dos trabalhos existentes utilizados, que a PBR é uma técnica eficiente para se encontrar defeitos em artefatos de *software* do que o método *Ad hoc*. Como a CBR já havia sido selecionada para ser utilizada no experimento documentado nessa monografia, a PBR foi a única outra técnica encontrada que se encaixou nos requisitos desse experimento. A PBR é utilizada para fazer uma análise baseada em perspectiva, e os *tickets* cadastrados pelos clientes na empresa onde foi utilizada precisam contemplar a perspectiva dos mesmos, por esse motivo a PBR foi selecionada.

2.6 Considerações finais sobre a revisão sistemática

A revisão sistemática realizada resultou em apenas 3 artigos selecionados de acordo com os critérios de inclusão e exclusão definidos na Seção 2.3.3. Muitos dos artigos encontrados que documentavam o uso das técnicas de inspeção de *software* procuradas eram focados em documentar sua utilização para analisar código-fonte ou diagramas UML, o que não é o foco do experimento documentado nesta monografia. A partir da leitura desses artigos, foi possível escolher as técnicas, documentadas na Seção 2.1, que foram utilizadas para a realização do experimento documentado nesta monografia. Tendo as técnicas utilizadas definidas, será explicitado no próximo capítulo como elas foram utilizadas e como as análises feitas utilizando os dados provenientes da empresa escolhida como caso de estudo foram conduzidas.

3 Estudo experimental

Nesse capítulo será explicado como a PBR e CBR foram utilizadas dentro da empresa selecionada, assim como o ciclo de vida de um *ticket*, os artefatos que foram analisados, como os dados de análise foram coletados e quais testes estatísticos foram utilizados para analisar os resultados obtidos. Essas técnicas não foram utilizadas anteriormente pela empresa e a mesma não tinha conhecimento sobre elas.

3.1 Ciclo de vida de um *ticket*

Antes de explicitar como a PBR e CBR foram utilizadas na organização selecionada para fazer parte desse estudo experimental, é necessário explicar o ciclo de vida de um *ticket* dentro dessa organização para entender em que momento do processo as técnicas de inspeção de *software* foram de fato utilizadas.

Na Figura 3.1 é possível observar o processo utilizado para analisar um *ticket* desde seu cadastro até seu fechamento. Quando um *ticket* é cadastrado, ele inicialmente é analisado por uma equipe de suporte ao produto, que é responsável por fazer uma análise não técnica do problema documentado no *ticket*. Caso essa equipe chegue à conclusão que o *ticket* de fato documenta um problema no sistema, ela o encaminha para a equipe de desenvolvedores, caso contrário, ela fecha o *ticket* dando um *feedback* para o cliente explicando o motivo do problema documentado não ser um defeito no sistema auxiliando-o a resolver sua dificuldade no uso do mesmo.

Quando o *ticket* é encaminhado para a equipe de desenvolvedores, é feita uma outra análise, dessa vez de cunho mais técnico, para verificar se o problema documentado de fato explicita um defeito no *software*. Caso a equipe de desenvolvedores chegue à conclusão que o *ticket* não documenta um defeito do sistema, o mesmo é retornado para a equipe de suporte ao produto para fazê-los compreender o motivo do retorno e para eles darem um *feedback* correto para o cliente. Nesse caso, é importante que a equipe de suporte ao produto analise o *ticket* novamente levando em consideração a análise feita

pelos desenvolvedores para poderem compreender com detalhes o problema que o cliente está enfrentando e como esse problema pode ser resolvido. Se a equipe de desenvolvedores decidir que o problema pelo qual o cliente está passando é de fato um defeito no *software*, a mesma corrige esse defeito e fecha o *ticket*.

Ainda na Figura 3.1, pode-se observar que é sinalizado que a etapa em que as técnicas de inspeção de *software* serão utilizadas é no momento onde o *ticket* é analisado pela equipe de desenvolvedores. Isso se deve ao fato de, logo no início do processo de análise do *ticket* pelos desenvolvedores, será feita uma análise para identificar se os problemas documentados representam de fato um defeito no *software*. O time de desenvolvedores conta com um analista de *tickets*, que é o profissional responsável por analisar os *tickets* antes de enviá-los para os desenvolvedores. Esse é o profissional que utilizou as técnicas de inspeção de *software* para identificar defeitos nos *tickets* e no *software*. A ideia de analisar o *ticket* antes de enviá-lo para os desenvolvedores visa garantir que, quando o desenvolvedor comece a trabalhar na análise técnica de um *ticket*, já tenha alguma garantia que ele de fato documenta um defeito no *software*, e não um comportamento esperado do mesmo.

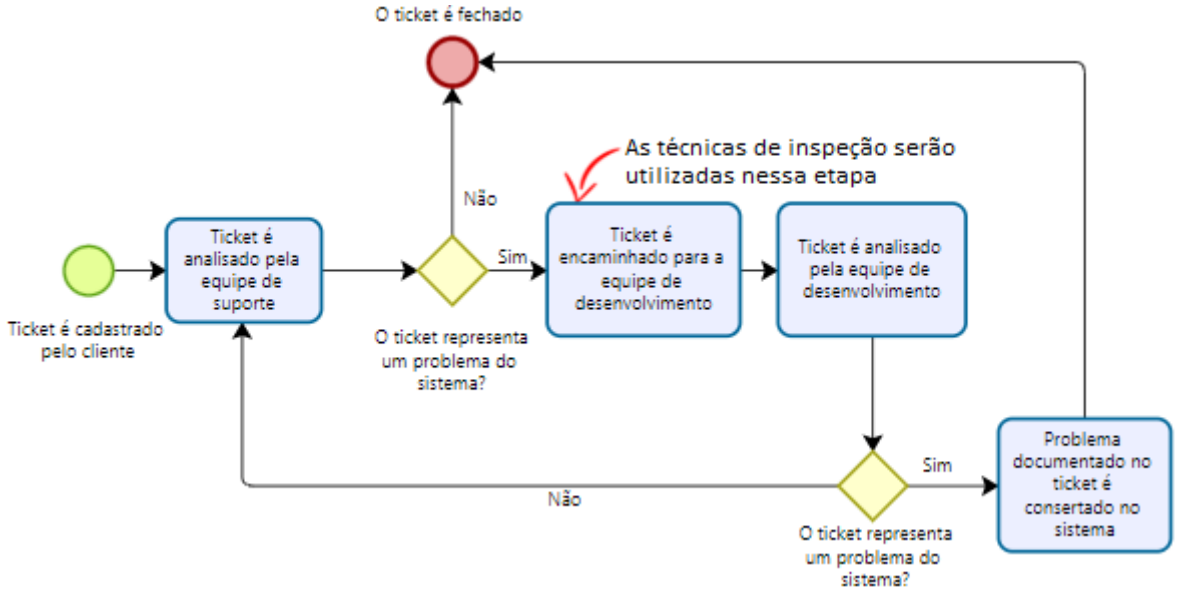

Figura 3.1: Processo de análise de um *ticket*

3.2 Utilização da PBR com a CBR

Os seguintes artefatos de *software* foram analisados utilizando a PBR e a CBR:

- *ticket* cadastrado pelo cliente;
- socumentos de *software*.

Nesse estudo experimental, a PBR foi utilizada para analisar os *tickets* cadastrados pelos clientes de acordo com a perspectiva deles e dos desenvolvedores. É necessário utilizar a perspectiva do cliente para entender a necessidade dele, e isso é feito através do cenário documentado por ele no *ticket* que registrou. A perspectiva do desenvolvedor também é importante pelo fato de ele ser a pessoa que resolverá o problema do cliente, portanto, é necessário que o *ticket* que será analisado contenha informações relevantes e necessárias para levar o desenvolvedor a realizar uma análise assertiva.

Entendendo a necessidade de se analisar *tickets* levando as duas perspectivas, do usuário e do desenvolvedor, em consideração, foram criadas *checklists*, que podem ser vistas no Apêndice A, com a finalidade de garantir que cada *ticket* cadastrado de fato documenta um defeito no *software* e que contém todas as informações relevantes para o desenvolvedor resolver o problema da melhor forma possível, garantindo a satisfação do cliente. Essas *checklists* visam garantir que a perspectiva do cliente esteja presente no *ticket* cadastrado e que o desenvolvedor tenha sua perspectiva contemplada quando todas as informações necessárias para a correção do problema reportado no *ticket* estejam presentes no mesmo.

Na organização utilizada como caso de estudo, é comum que problemas recorrentes sejam encaminhados para a equipe de desenvolvedores. Uma parte desses problemas geralmente não são considerados defeitos, mas sim um comportamento correto do sistema. Para auxiliar na análise desses problemas que são recorrentes, foram criadas outras *checklists*. Com as *checklists* criadas para auxiliar na análise desses problemas, isso tende a facilitar a inspeção feita pelo analista de *tickets* tendo como perspectiva a visão do desenvolvedor.

Os documentos de *software* foram analisados de acordo com as análises dos *tickets*. Como cada *ticket* deve documentar um defeito no *software*, é necessário que esses

documentos sejam analisados de acordo com a funcionalidade que está sendo reportada no *ticket*. Esse processo visa garantir a análise de *tickets* levando em consideração a especificação dos requisitos da funcionalidade reportada nos mesmos. Essa etapa é de extrema importância, pois é nela que será decidido de fato se o *ticket* documenta um defeito no *software*.

3.3 Classificações dos *Tickets*

Para entender como os resultados foram analisados, é necessário explicar como os *Tickets* retornados pela equipe de desenvolvimento são classificados quando eles não documentam um defeito no *software*. Eles são classificados da seguinte forma:

- *as designed*: o *ticket* classificado dessa forma não documenta um problema no *software*, mas sim um comportamento esperado de acordo com a documentação;
- *deferred*: representa o *tickets* que de fato documenta um problema no *software*, mas um problema que já é conhecido, mapeado, esperado e que não existe uma solução para ele ainda;
- *cannot reproduce*: classificação dada a um *ticket* que documenta um problema que não foi possível de ser reproduzido pela equipe de desenvolvedores.

Além dessas classificações, foi analisado também o total de *tickets* retornados pela equipe de desenvolvimento, que se trata da soma de todos os *ticket* retornados utilizando as classificações explicitadas.

Fazendo um paralelo às classificações especificadas na Seção 2.1.1, o *ticket* classificado como *as designed* pode representar os itens que possuem defeitos que contém fatos incorretos, pois a especificação errada de fatos na documentação do *ticket* pode levar ele a não ser considerado um defeito, mas sim um comportamento esperado. O item classificado como *cannot reproduce* pode representar os *tickets* que possuem defeitos ambíguos, pelo fato da ambiguidade atrapalhar na reprodução do problema do cliente; inconsistentes, pelo fato da inconsistência de escrita do *ticket* atrapalhar na análise pelos desenvolvedores; ou omissivos, pois informações omissas no *ticket* podem levar o desenvolvedor a não

conseguir reproduzir o defeito relatado pelo cliente no *software*. Não foi possível fazer um paralelo com os *tickets* classificados como *deferred*, pelo fato dessa classificação de fato documentar um problema no *software*. Apesar de *tickets* classificados dessa forma de fato documentarem um problema no sistema, eles são tratados da mesma forma que itens classificados como *as designed* e *cannot reproduce*, pois se tratam de problemas que não tem solução e que já são mapeados, portanto, conhecidos pela equipe de suporte.

As classificações explicadas nessa seção foram utilizadas para o estudo dos resultados desse trabalho pois a empresa usada como caso de estudo preferiu utilizar as classificações já existentes nela ao invés das criadas por Shull (1998), que foram explicadas na Seção 2.1.1 desse trabalho. Mesmo com essa preferência da empresa, foi possível fazer um paralelo entre as classificações pois os tipos de defeitos existentes eram os mesmos.

3.4 Dados usados para análise

Para fazer a análise da influência da implantação das técnicas de inspeção de *software* nessa empresa, foram utilizadas duas métricas necessárias para se cumprir o objetivo especificado na Seção 1.1.2, que se trata da verificação do impacto da utilização dessas técnicas em um cenário onde elas não foram utilizadas com um cenário onde foram.

A primeira métrica é a quantidade de *tickets* que foram reanalisados pela equipe de desenvolvedores antes e depois da implantação da PBR e CBR. A segunda métrica é a quantidade de *tickets* que os desenvolvedores encaminharam para a equipe de suporte alegando que os mesmos documentavam um comportamento correto do sistema antes da utilização da PBR e CBR e quantos *tickets* foram encaminhados após a implantação dessas técnicas. Os dados necessários para realizar esse estudo experimental, como a quantidade de *tickets* que foram fechados antes e depois da implantação da PBR e CBR, foram fornecidos pela empresa que está sendo utilizada como caso de estudo.

As análises foram feitas utilizando cada classificação especificada na seção anterior de forma separada para verificar o impacto da implantação das técnicas CBR e PBR em cada uma delas. Foi feito um levantamento sobre a quantidade mensal de *tickets* retornados para a equipe de suporte e sobre a classificação de cada um deles. Com esses

dados, foi calculada a porcentagem de *tickets* que cada classificação possuía em relação à quantidade total do mês, e toda a análise foi feita em utilizando essas porcentagens. Foram utilizados os dados referentes ao período de Janeiro de 2021 a Janeiro de 2022, onde de Janeiro de 2021 a Outubro de 2021 representa o grupo de *tickets* que não foram analisados utilizando a CBR e PBR, enquanto os dados referentes a Novembro de 2021 a Janeiro de 2022 representa o grupo de *tickets* que foram analisados utilizando as técnicas. Portanto, foram utilizados os seguintes grupos de dados:

- G1: Grupo de dados referente ao período onde as técnicas não foram utilizadas;
- G2: Grupo de dados referente ao período onde as técnicas foram utilizadas.

Além desses grupos, foram utilizadas quatro variáveis diferentes, que são as classificações *As Designed*, *Cannot Reproduce*, *Deferred* e "total de *tickets* criados", especificadas na Seção 3.3. O conjunto de dados utilizados, que foi fornecido pela empresa utilizada por esse experimento, pode ser visualizado na Tabela 3.1.

Tabela 3.1: Dados utilizados para a realização das análises estatísticas.

Grupos	*As Designed*(%)	*Cannot reproduce*(%)	*Deferred*(%)	Total recusados(%)
G1	10,93	3,64	4,86	19,43
G1	22,65	2,21	9,94	34,81
G1	23,77	5,74	9,02	38,52
G1	20,99	7,41	13,58	41,98
G1	10,24	8,66	5,51	24,41
G1	8,89	7,41	4,44	20,74
G1	6,21	1,38	6,21	13,79
G1	10,32	7,10	4,52	21,94
G1	9,87	3,29	3,95	17,11
G1	7,75	3,10	10,08	20,93
G2	13,60	0,80	3,20	17,60
G2	12,00	4,00	6,00	22,00
G2	6,67	3,33	1,67	11,67

3.5 Testes estatísticos utilizados

Foi feita uma análise estatística utilizando os resultados para avaliar o impacto que a implantação das técnicas teve no *backlog* do time de desenvolvimento e verificar se houve alguma diferença significativa. Para fazer essa análise, foi utilizado o programa Minitab[2].

[2]Acessível através de ⟨https://www.minitab.com/pt-br/⟩

Através dele foi possível fazer testes de normalidade, homocedasticidade e testes não-paramétricos. Para realizar esses testes, foram feitos testes de hipóteses, que visam tomar uma decisão escolhendo entre duas ou mais hipóteses definidas (FISHER, 1992), utilizando os dados obtidos antes e após o experimento ter sido feito. Essas hipóteses são importantes para avaliar o resultado obtido no experimento e dizer se ele de fato pode ter tido um impacto positivo. Para avaliá-las, foi utilizado um nível de significância de 5%, que se trata da probabilidade de não ter obtido um resultado positivo com o experimento feito (SALKIND, 2006).

O teste de normalidade utilizado segue o método definido por Ryan (1976), chamado de Ryan-Joiner. Para realizar o teste não-paramétrico foi utilizado o método de Mann-Whitney, definido por Mann e Whitney (1947), que visa testar duas amostras independentes e que podem possuir diferentes tamanhos. Para realizar o teste de homocedasticidade, foi utilizado o método de Levene, descrito por Brown e Forsythe (1974). O teste T para duas variâncias utilizado foi definido por Connelly (2011). Todos os testes foram realizados através do Minitab.

O primeiro passo feito para analisar todas as classificações foi realizar um teste de normalidade utilizando o método de Mann-Whitney através do Minitab. Para isso, foram utilizadas duas hipóteses:

- NH0: os dados possuem uma distribuição normal;
- NH1: os dados não possuem uma distribuição normal.

Após essa etapa, caso os dados possuam uma distribuição normal(hipótese HN0 verdadeira), deve ser feito um teste de homocedasticidade. Esse teste visa definir se as variâncias dos grupos de dados são homogêneas ou não, e ele só deve ser executado caso os dados possuam uma distribuição normal (BROWN; FORSYTHE, 1974). Para esse tipo de teste, foram utilizadas as seguintes hipóteses:

- HH0: os dados são homocedásticos;
- HH1: os dados não são homocedásticos.

Após a execução dos testes de homocedasticidade, caso a hipótese HH0 seja verdadeira, deve ser feito o Teste T, que tem como base a comparação das médias dos grupos

de dados que estão sendo analisados (CONNELLY, 2011). Esse teste só deve ser realizado caso os dados sigam uma distribuição normal, ou seja, tenham a hipótese NH0 como verdadeira, e tenham uma variância homogenia (CONNELLY, 2011). Para a realização desse teste, também são utilizadas duas hipóteses:

- TH0: não houve diferença significativa;
- TH1: houve diferença significativa.

Tendo os testes de hipóteses definidos, os resultados obtidos após a execução deles estarão documentados no próximo capítulo.

4 Resultados

Nesse capítulo será abordada a discussão sobre os resultados obtidos do experimento de acordo com a análise estatística feita.

4.1 Variável *As Designed*

O teste de normalidade de Ryan-Joiner foi executado no Minitab utilizando a classificação *As Designed* e o gráfico exibido na Figura 4.1 foi gerado. Analisando essa figura, é possível observar que o Valor-P foi calculado pelo MiniTab. Esse valor é utilizado para dizer qual é a probabilidade da hipótese NH0 ser verdadeira quando obtemos o padrão de resultados que foi encontrado no experimento. Como foi utilizado um nível de significância de 5% que limita a probabilidade máxima para que a hipótese NH0 seja considerada como verdadeira, pode-se chegar à conclusão que os dados não possuem uma distribuição normal pelo fato do Valor-P ser menor do que o nível de significância estipulado, portanto, assume-se a hipótese NH1 como verdadeira nesse caso.

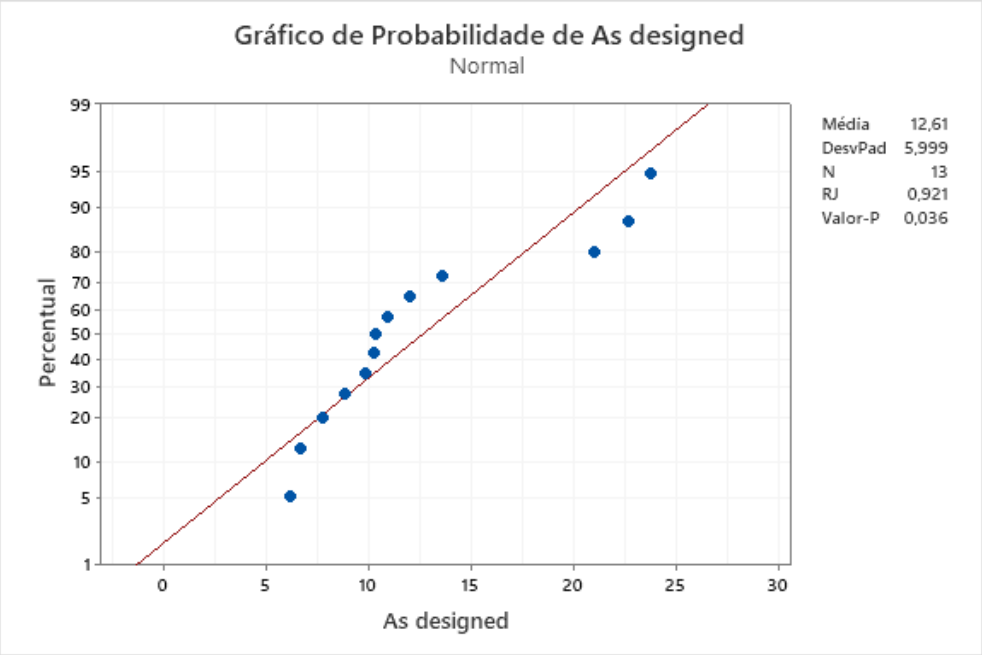

Figura 4.1: Gráfico de Probabilidade da variável *As Designed* para verificar normalidade. Gráfico gerado pelo Minitab.

Chegando à conclusão que os dados não possuem uma distribuição normal, foi feito o teste não-paramétrico de Mann-Whitney. Para realizar esse teste, foram definidas duas hipóteses:

- NPH0: não houve diferença significativa;
- NPH1: houve diferença significativa.

Observando o resultado gerado pelo teste de Mann-Whitney na Figura 4.2, é possível verificar que o Valor-P possui um valor maior do que o nível de significância estipulado de 5%, portanto, a hipótese NPH0 é aceita como verdadeira. Tendo o Valor-P igual a 1 significa que há 100% de chance da hipótese NPH0 ser verdadeira. Através dessa análise estatística é possível constatar que não há uma diferença significativa entre os dados obtidos antes de aplicar as técnicas com os dados obtidos após a aplicação das mesmas. Portanto, constata-se que as técnicas utilizadas não foram eficientes para identificar defeitos decorrentes de fatos incorretos.

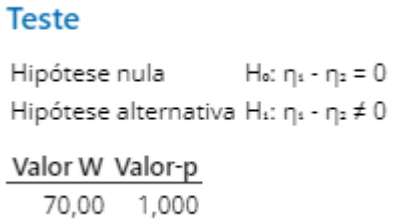

Figura 4.2: Resultado do teste não-paramétrico da variável *As Designed*. Imagem gerada pelo Minitab.

4.2 Variável *Cannot Reproduce*

A primeira etapa realizada para avaliar os *tickets* classificados como *Cannot Reproduce* foi realizar o teste de normalidade dos dados obtidos. Na Figura 4.3 pode ser observado que o Valor-P obtido pelo teste é maior do que o nível de significância estabelecido de 5%, portanto, podemos considerar a hipótese NH0 como verdadeira.

Tendo a hipótese NH0 como verdadeira, foi feito um teste de homocedasticidade para verificar a variância dos dados. Na Figura 4.4 é possível observar que o Minitab

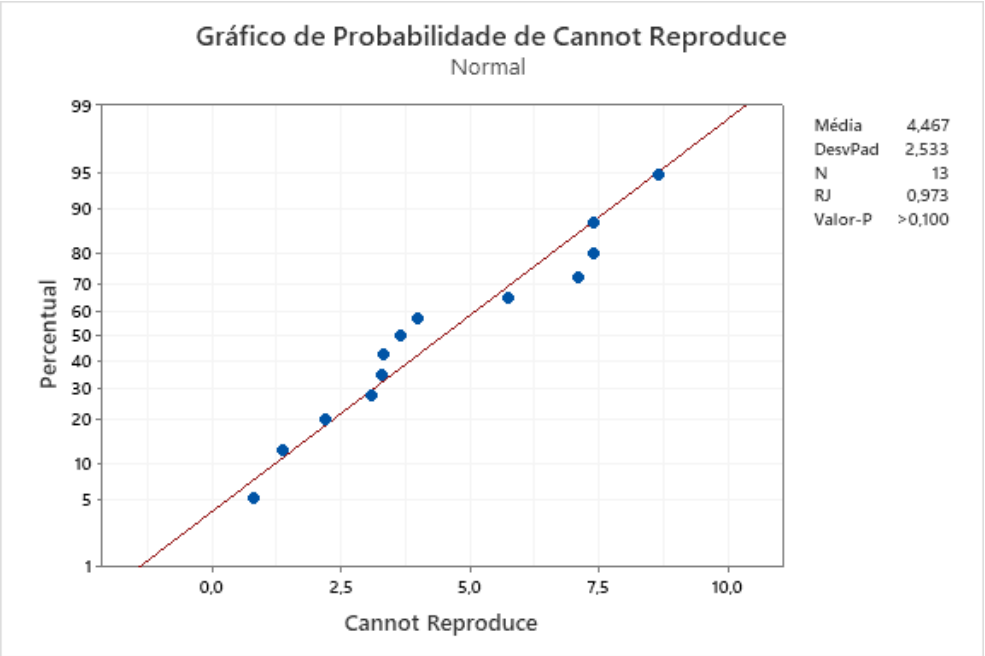

Figura 4.3: Resultado do teste de normalidade da variável *Cannot Reproduce*. Imagem gerada pelo Minitab.

realiza dois testes, o de Bonett e o de Levene. Nesse estudo foi considerado o teste de Levene pois este é utilizado para avaliar grupos de dados que possuem uma amostragem menor do que 20. Nessa figura, é explicitado o Valor-P obtido após a execução do teste de Levene. Como o Valor-P de 0,109 é maior do que o nível de significância estipulado de 5%, pode-se assumir a hipótese HH0 como verdadeira e afirmar que os dados são homocedásticos.

Como os dados seguem uma distribuição normal e são homocedásticos, foi feito o Teste T para verificar se de fato há alguma diferença significativa entre os dois conjuntos de dados analisados. O resultado da execução desse teste pode ser observado na Figura 4.5. A parte importante desse resultado é o Valor-P, que é igual a 0,132. Como o Valor-P é maior do que o nível de significância de 5%, a hipótese TH0 pode ser aceita, pois a probabilidade dela ser verdadeira ultrapassa o limite estipulado pelo nível de significância.

Fazendo uma análise geral sobre o estudo dessa variável, de acordo com os métodos estatísticos realizados, não há uma diferença significativa entre os dois grupos de dados, mas é possível observar no terceiro *bloxpot* da Figura 4.4 que a mediana entre os dois grupo diminuiu. Isso poderia ser um bom indício de melhoria, mas que não pode ser avaliado de forma isolada.

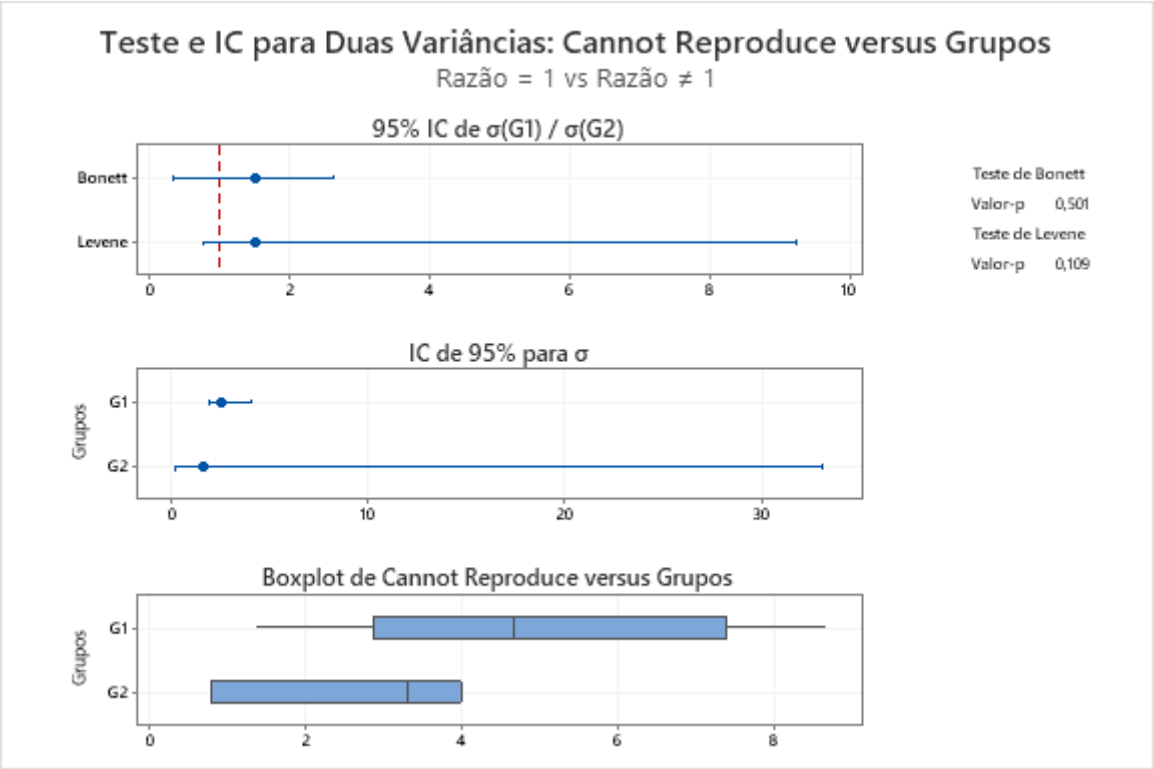

Figura 4.4: Resultado do teste de homocedasticidade da variável *Cannot Reproduce*. Imagem gerada pelo Minitab.

Teste

Hipótese nula	H_0: $\mu_1 - \mu_2 = 0$
Hipótese alternativa	H_1: $\mu_1 - \mu_2 \neq 0$

Valor-T	GL	Valor-p
1,80	5	0,132

Figura 4.5: Resultado do Teste T da variável *Cannot Reproduce*. Imagem gerada pelo Minitab.

4.3 Variável *Deferred*

O roteiro de testes utilizado para avaliar o impacto do uso das técnicas com *tickets* de classificação *Deferred* é o mesmo utilizado na variável *Cannot Reproduce*. Essa variável também possui uma distribuição de dados normal, tendo a hipótese NH0 como verdadeira no teste de normalidade, e os dados também podem ser considerados homocedásticos, tendo a hipótese HH0 como verdadeira no teste de homocedasticidade.

Tendo os resultados do teste de normalidade na Figura 4.6, é possível verificar que o Valor-P é maior que o nível de significância de 5%, por isso podemos assumir a

Figura 4.6: Resultado do teste de normalidade da variável *Deferred*. Imagem gerada pelo Minitab.

hipótese NH0 como verdadeira. Já com resultado do teste de homocedasticidade exibido na Figura 4.7, pode-se observar que o Valor-P, obtido pelo teste de Levene, também é maior que o nível de significância, portanto assume-se que a hipótese HH0 é verdadeira.

Como os dados relacionados à essa variável são considerados normais e homocedásticos, podemos prosseguir para o Teste T para avaliar se houve alguma diferença significativa na utilização das técnicas. Conforme observado na Figura 4.8, pode-se concluir que a hipótese TH0 é tida como verdadeira pelo fato do Valor-P ser maior que o nível de significância de 5%.

Apesar de não ser observada uma diferença significativa analisando apenas a variável *Deferred* isoladamente, é possível observar que as técnicas tiveram um impacto mais positivo nela do que na variável *Cannot Repoduce*. Quando é feita uma comparação entre o Valor-P obtido pelo Teste T das duas variáveis, pode-se verificar que o mesmo possui um valor menor quando é referente à variável *Deferred*. Enquanto o Teste T da variável *Deferred* retornou uma probabilidade de 0,093(Valor-P) da hipótese TH0 ser verdadeira, o mesmo teste para a variável *Cannot Repoduce* retornou uma probabilidade de 0,132 para a mesma hipótese ser verdadeira. Além disso, analisando o terceiro *bloxpot* exibido na Figura 4.7, pode-se observar que há uma diferença maior entre as medianas

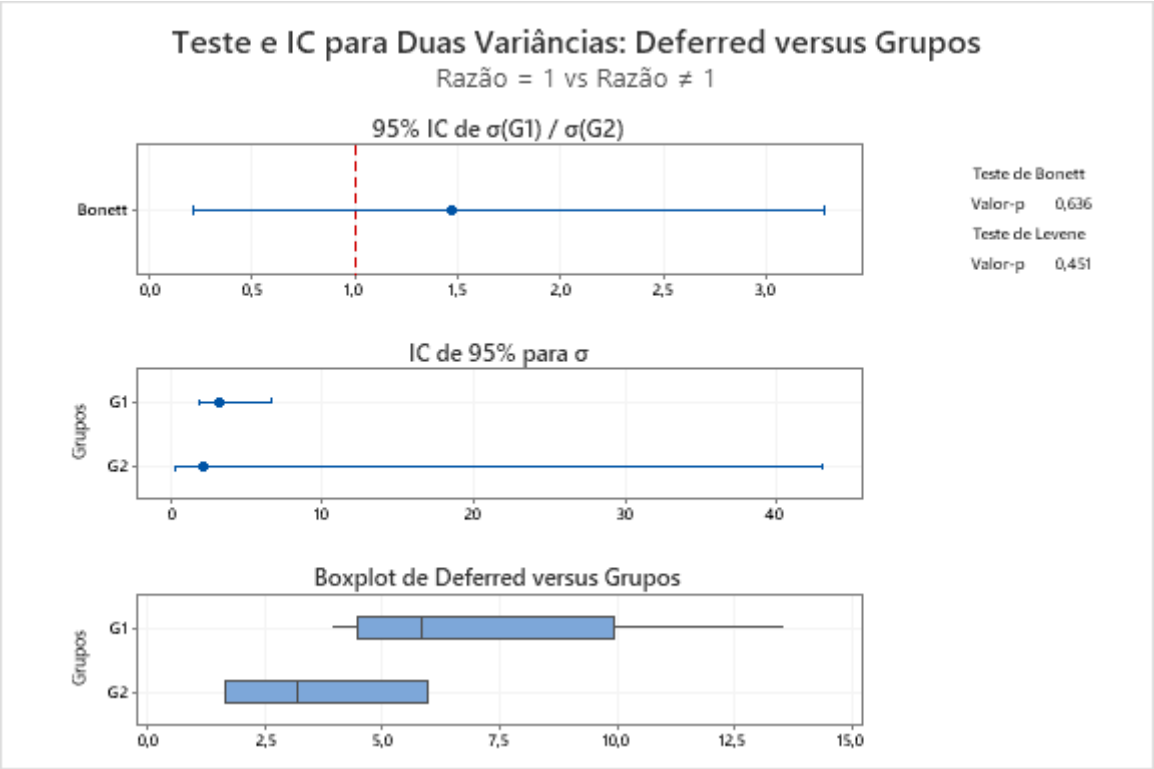

Figura 4.7: Resultado do teste de homocedasticidade da variável *Deferred*. Imagem gerada pelo Minitab.

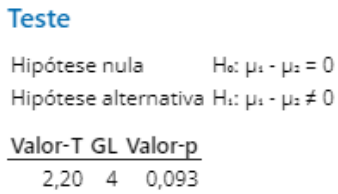

Figura 4.8: Resultado do Teste T da variável *Deferred*. Imagem gerada pelo Minitab.

dos grupos de dados da variável *Deferred* do que o observado na Figura 4.4 relacionado aos grupos de dados da variável *Cannot Reproduce*.

4.4 Variável "Total de *tickets* recusados"

A análise da variável "Total de *tickets* recusado" é importante para verificar se a utilização das técnicas de inspeção teve algum impacto relevante no total de *tickets* recusados, mesmo que não tenha tido uma diferença significativa nas outras classificações. A análise dessa variável seguiu da mesma forma que as duas últimas(*Cannot Reproduce* e *Deferred*),

fazendo um teste de normalidade e constatando que os dados podem ser considerados normais, e fazendo um teste de homocedasticidade e constatando que os dados possuem uma variância homogênea.

Através da Figura 4.9 é possível observar que o Valor-P(0,079) encontrado pelo teste de normalidade é maior que o nível de significância de 5%, portanto, podemos assumir a hipótese NH0 como verdadeira. Como os dados possuem uma distribuição normal, foi feito o teste de homocedasticidade, seu resultado pode ser observado na Figura 4.10. Como o Valor-P de 0,452 obtido pelo teste de Levene é maior que o nível de significância de 5%, é aceita a hipótese HH0 como verdadeira, onde os dados podem ser considerados homocedásticos.

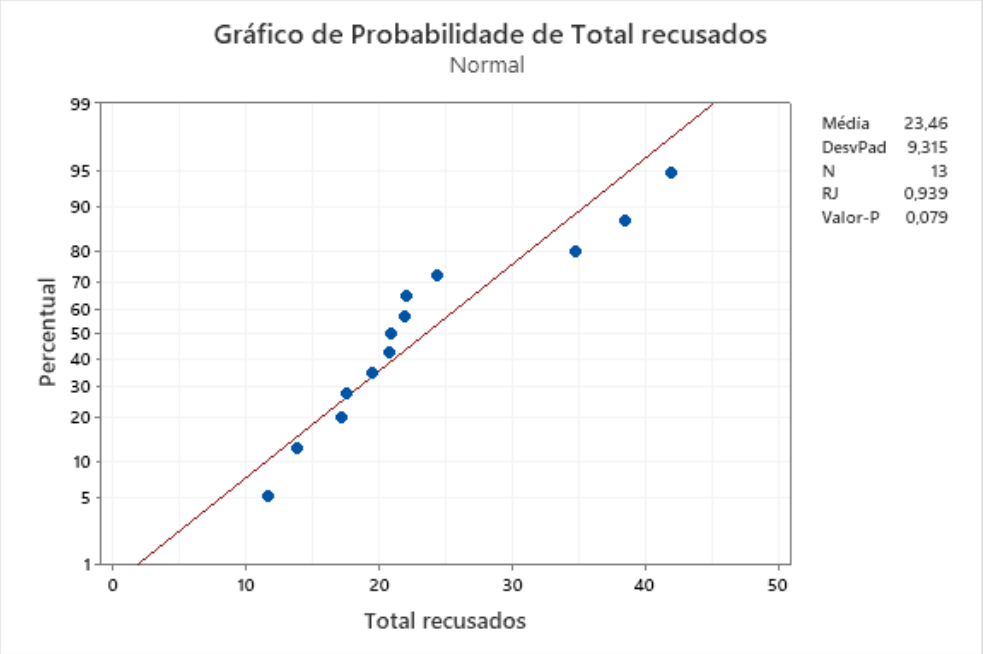

Figura 4.9: Resultado do teste de normalidade da variável “Total de *tickets* recusados”. Imagem gerada pelo Minitab.

Tendo os dados com uma distribuição normal e possuindo uma variância homocedástica, foi feito o Teste T dessa variável. O resultado desse teste pode ser visualizado na Figura 4.11, onde foi obtido um Valor-P de 0,1, que representa uma probabilidade maior que o limite estipulado pelo nível de significância. Como a probabilidade da hipótese TH0 ser verdadeira é maior do que o valor estipulado pelo nível de significância, ela foi aceita como verdadeira, onde é considerado que não há uma diferença significativa entre os conjuntos de dados analisados.

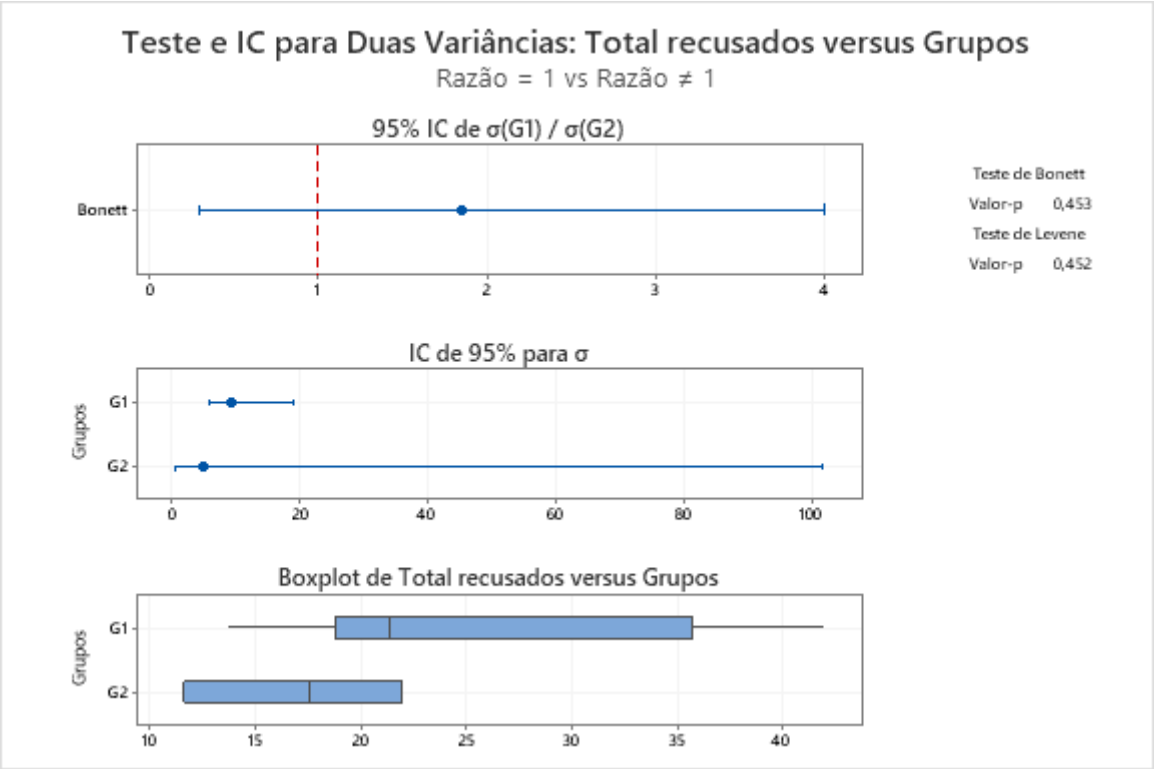

Figura 4.10: Resultado do teste de homocedasticidade da variável "Total de *tickets* recusados". Imagem gerada pelo Minitab.

Fazendo uma comparação dos resultados obtidos nos Testes T dessa variável com as demais, é possível observar que o Valor-P dela foi maior que o da variável *Deferred*, mas menor que da variável *Cannot Reproduce*, o que indica que a probabilidade da hipótese TH0 ser verdadeira está menor quando comparada com a variável *Deferred*, mas maior quando comparada com a variável *Cannot Reproduce*.

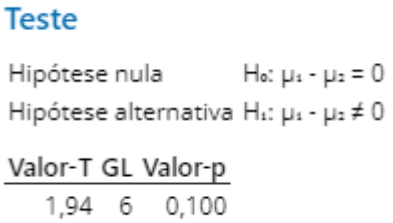

Figura 4.11: Resultado do Teste T da variável "Total de *tickets* recusados". Imagem gerada pelo Minitab.

4.5 Discussão

Para compreender se o objetivo específico explicitado na Seção 1.1.2 pôde ser atingido através do uso das técnicas de inspeção de *software* selecionadas, foi feita a análise estatística apresentada no presente capítulo. O resultado bruto das análises feitas explicita que, durante um período de três meses de utilização da CBR juntamente com a PBR, não foi possível observar uma melhoria significativa na quantidade de itens cadastrados no *backlog* do time de desenvolvimento utilizado como caso de estudo quando os *tickets* seguem as classificações explicitadas na Seção 3.3. Mas, algumas discussões podem ser feitas a partir desse resultado.

A primeira discussão que pode ser feita é sobre a qualidade das *checklists* utilizadas. Como o resultado obtido num período de três meses não mostrou uma diferença significativa no *backlog* do time, foi feito um questionamento se o método de análise utilizado anteriormente já era bom o suficiente, e, por isso, o uso das *checklists* não surtiu efeito de imediato. Em uma análise das *checklists* criadas juntamente com o analista que as utilizou, foi possível verificar que elas de fato contemplavam todos os dados necessários para um desenvolvedor prosseguir com a análise técnica necessária. Não existia um método formalizado para a análise de *tickets* anteriormente na equipe de desenvolvimento onde as *checklists* foram utilizadas e, apesar dos resultados não terem expressado uma diferença significativa, ter uma forma de análise padronizada ajudou a equipe de suporte a ter mais visibilidade de quais dados são necessários para uma equipe de desenvolvimento analisar um *ticket*. As *checklists* poderiam ser melhoradas caso o *software* utilizado tivesse uma documentação mais completa.

Uma outra discussão que pode ser aberta, é sobre o período de três meses que as *checklists* foram utilizadas. Foi feito um questionamento se um período de três meses é o suficiente para obter dados necessários para se realizar uma análise estatística relevante que demonstre bons resultados nesse estudo. Esse de fato pode ter sido um problema na implantação das técnicas. Observando os dados brutos, antes da análise estatística feita, nos primeiros dois meses de aplicação das técnicas, é possível verificar que a quantidade de *tickets* que estavam sendo retornados para a equipe de suporte não diminuiu. Essa diminuição só ocorreu no terceiro mês de aplicação das técnicas, que foi em Janeiro de

2022. Talvez o uso dessas técnicas possa levar tempo para surtir um efeito positivo onde elas estão sendo implementadas, porque não era utilizado nenhum processo de inspeção formalizado anteriormente nesse time de desenvolvimento. Além disso, uma maior quantidade de dados seria ideal para fazer as análises estatísticas que foram realizadas nesse estudo, e só seria possível obter esses dados caso o experimento continuasse durante alguns meses a mais.

A última discussão feita é sobre os defeitos que não foram possíveis de ser identificados nos *tickets* através do uso das técnicas escolhidas. Fazendo uma análise dos *tickets* que continuam sendo recusados pelo analista que trabalha na equipe de desenvolvedores, é possível verificar que a maioria deles possuem um problema em comum. Muitos dos que foram recusados tentam apresentar defeitos em funcionalidades que possuem pouca, ou nenhuma especificação. A falta de documentação de algumas funcionalidades do *software* pode impactar negativamente no trabalho de um time de desenvolvimento. A inexistência de uma especificação para algumas funcionalidades do *software* impactou diretamente na utilização das técnicas PBR e CBR.

5 Conclusão

O uso de técnicas das inspeção de *software* para identificar defeitos nos artefatos do mesmo é uma prática conhecidamente positiva, referência em diversos estudos, como os citados no Capítulo 2. Os artefatos analisados podem ser de diversas naturezas diferentes, sejam eles códigos-fonte do produto, ou até mesmo uma documentação escrita em linguagem natural. Nesse estudo, as técnicas PBR e CBR foram utilizadas visando analisar os *tickets* de uma empresa de desenvolvimento de *software*, com o objetivo de verificar se a utilização dessas técnicas pode diminuir o *backlog* de um time que adota uma metodologia ágil de desenvolvimento.

A utilização de algumas técnicas de inspeção em times que adotam uma metodologia ágil de desenvolvimento pode se tornar um desafio, já que a uso delas não deve impactar no fluxo de desenvolvimento dos mesmos de forma negativa. Por causa desse motivo, foram selecionadas técnicas para realizar esse experimento que podem ser utilizadas por apenas um indivíduo e que não impactariam nas entregas do time onde foram implantadas.

Através das análises dos resultados obtidos a partir dos testes de hipótese feitos, foi possível observar que não houve um impacto significativo no *backlog* do time de desenvolvimento usado como caso de estudo para o presente trabalho. Apesar do impacto não ter sido significativo, foi possível verificar que os *tickets* que ainda estavam sendo recusados pelo analista da equipe de desenvolvimento, documentavam possíveis defeitos em funcionalidades que não têm seus requisitos totalmente, ou parcialmente, documentados. A falta dessa documentação pode ter impactado de forma negativa no uso e criação das *checklists* utilizadas para analisar os *tickets* dessa empresa. Além disso, a falta dessa documentação está associada ao fato da empresa priorizar ter o *software* em funcionamento do que a documentação do mesmo completa, que é uma característica do desenvolvimento ágil, o que talvez seja um indício que a utilização das técnicas selecionadas não sejam ideais para serem utilizadas em times que utilizam uma metodologia ágil de desenvolvimento.

De acordo com o manifesto ágil[3], é prioritário ter um *software* em funcionamento do que uma documentação completa. Como as *checklists* criadas visam orientar o analista de *tickets* a fazer uma inspeção levando em consideração a documentação existente do *software*, a falta de uma documentação completa foi impactante para o uso das técnicas selecionadas em um time que utiliza uma metodologia ágil de desenvolvimento.

Tendo em vista os aspectos apresentados, analisando o objetivo desse trabalho, é possível concluir que, dentro de um período de três meses, não foi possível observar uma melhoria significativa na diminuição do *backlog* do time de desenvolvimento escolhido quando se trata os *tickets* utilizando as técnicas selecionadas. Com o problema da falta de documentação identificado, tendo como origem a metodologia ágil de desenvolvimento adotada no time, pode-se dizer que de acordo com os dados coletados e análises realizadas, não foram encontradas evidências que a utilização das técnicas foram eficientes ao serem utilizadas em um time de desenvolvimento de *software* que adota uma metodologia ágil de desenvolvimento. Como sugestão para trabalhos futuros, podem ser realizados estudos experimentais que verificam se para utilizar a PBR juntamente com a CBR é necessário que o *software* tenha uma documentação completa. Todos os artigos escolhidos através da revisão sistemática documentam a utilização dessas técnicas em um cenário onde a documentação do *software* é existente e completa, por isso seria interessante existirem estudos comparando os resultados da utilização das técnicas em um cenário onde a documentação do *software* não esteja completa com um cenário onde ela esteja. Dessa forma, seria possível obter conclusões concretas se a falta de documentação de fato prejudica na utilização das técnicas de inspeção de *software* PBR e CBR.

[3]Acessível através de ⟨https://agilemanifesto.org/iso/ptbr/manifesto.html⟩

Bibliografia

BIFFL, S. Analysis of the impact of reading technique and inspector capability on individual inspection performance. In: IEEE. *Proceedings Seventh Asia-Pacific Software Engeering Conference. APSEC 2000.* [S.l.], 2000. p. 136–145.

BROWN, M. B.; FORSYTHE, A. B. Robust tests for the equality of variances. *Journal of the American Statistical Association*, Taylor & Francis, v. 69, n. 346, p. 364–367, 1974.

BUCKLEY, J.; EXTON, C. Bloom's taxonomy: a framework for assessing programmers' knowledge of software systems. In: *11th IEEE International Workshop on Program Comprehension, 2003.* [S.l.: s.n.], 2003. p. 165–174.

CALDIERA, V. R. B. G.; ROMBACH, H. D. The goal question metric approach. *Encyclopedia of software engineering*, p. 528–532, 1994.

CIOLKOWSKI, M. What do we know about perspective-based reading? an approach for quantitative aggregation in software engineering. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement.* [S.l.: s.n.], 2009. p. 133–144.

CIOLKOWSKI, M.; LAITENBERGER, O.; BIFFL, S. Software reviews, the state of the practice. *IEEE Software*, v. 20, n. 6, p. 46–51, 2003.

CONNELLY, L. M. T-tests. *Medsurg Nursing*, Anthony J. Jannetti, Inc., v. 20, n. 6, p. 341, 2011.

DERMEVAL, D.; COELHO, J. A. d. M.; BITTENCOURT, I. I. Mapeamento sistemático e revisao sistemática da literatura em informática na educaçao. *JAQUES, Patrícia Augustin; PIMENTEL, Mariano; SIQUEIRA; Sean; BITTENCOURT, Ig.(Org.) Metodologia de Pesquisa em Informática na Educação: Abordagem Quantitativa de Pesquisa. Porto Alegre: SBC*, 2019.

FAGAN, M. E. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, v. 15, n. 3, p. 182–211, 1976.

FISHER, R. A. Statistical methods for research workers. In: _____. *Breakthroughs in Statistics: Methodology and Distribution.* New York, NY: Springer New York, 1992. p. 66–70. ISBN 978-1-4612-4380-9. Disponível em: ⟨https://doi.org/10.1007/978-1-4612-4380-9_6⟩.

GARCIA, T. M.; SILVA, M. G. G. d.; NASCIMENTO, R. P. C. d. Mapeamento sistemático: Adoção de padrões de interoperabilidade no governo. 2018. Disponível em: ⟨https://www.unirios.edu.br/revistarios/internas/conteudo/resumo.php?id=360⟩.

GAZERANI, N. et al. Developing a teaching framework to support software inspection. In: *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI).* [S.l.: s.n.], 2015. p. 84–90.

JOHNSON, P. M. Does every inspection really need a meeting. *Empirical Software Engineering*, p. 9–35, 1998.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

MANN, H. B.; WHITNEY, D. R. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, v. 18, p. 50–60, 1947.

MCMEEKIN, D. A.; KONSKY, B. R. von; CHANG, E. Checklist inspections and modifications: Applying bloom's taxonomy to categorise developer comprehension. In: *2008 16th IEEE International Conference on Program Comprehension*. [S.l.: s.n.], 2008. p. 224–229.

PETTICREW, M.; ROBERTS, H. *Systematic reviews in the social sciences: A practical guide*. [S.l.]: John Wiley & Sons, 2008.

RYAN, T. A. *Minitab student handbook*. [S.l.: s.n.], 1976.

SALKIND, N. J. *Encyclopedia of measurement and statistics*. [S.l.]: SAGE publications, 2006. 889–891 p.

SCHWABER, K.; BEEDLE, M. *Agile software development with Scrum*. [S.l.]: Prentice Hall Upper Saddle River, 2002. v. 1.

SHULL, F. J. *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. [S.l.], 1998.

APÊNDICE

A - *Checklists* utilizadas para realizar a inspeção

A.1 *Checklist* para defeitos comuns

O ticket contém as informações do escritório do cliente? Informações necessárias: URL, *tenancy*, senha e validade do usuário Support.

O ticket contém os steps para a reprodução do problema?

O ticket contém evidências do problema?

As configurações relacionadas ao problema já foram verificadas?

O ticket contém o resultado encontrado pelo cliente?

O ticket contém o resultado esperado da feature?

A regra de negócio para a feature defeituosa está especificada? (Essa pergunta não exclui o *ticket* caso a resposta seja "não")

O resultado esperado pelo sistema condiz com a especificação, caso ela exista?

A.2 *Checklist* para problemas de performance

O ticket contém as informações do escritório do cliente? Informações necessárias: URL, tenancy, senha e validade do usuário Support.

O ticket contém os steps para a reprodução do problema?

O ticket contém evidências do problema?

O ticket contém o resultado encontrado pelo cliente?

O ticket contém o resultado esperado da feature?

O ticket contém informações sobre o módulo em que ocorreu a lentidão?

O ticket contém a média de segundos ou minutos que a execução demorou?

O ticket contém a velocidade da internet do cliente?

O tipo de rede(wi-fi ou cabo) do cliente é especificada?

O ticket contém prints de teste de velocidade da internet do cliente?

O ticket contém o print do devtools, evidenciando que as requisições de fato estão lentas?

O roteador utilizado foi disponibilizado/instalado pelo provedor ou adquirido/instalado por serviço especializado?

É informado no ticket a data e o horário em que o problema aconteceu?